Instituto Tecnológico y de Estudios Superiores de Monterrey

Campus Monterrey

CS5051 Computational Techniques for Machine Learning

Final Project

Hyper-Heuristics Powered by Transformers for Customising Population-based Metaheuristics in Continuous Optimisation Problems

José Manuel Tapia Avitia
A00834191

Dr. César Torres Huitzil

November 24th, 2022

Contents

CS5051: Computational Techniques in Machine Learning
Final Project: Hyper-Heuristics Powered by Transformers for Customising Population-based Metaheuristics in Continuous Optimisation Problems

1 Abstract

Metaheuristics (MHs) are proven powerful algorithms for solving non-linear optimisation problems over discrete, continuous, or mixed domains. Applications have ranged from basic sciences to applied technologies. Nowadays, the literature contains plenty of MHs based on exceptional ideas, but they often combine elements from other techniques. An alternative approach is to follow a standard model that customises population-based MHs, utilising simple heuristics extracted from well-known MHs. Different approaches based on Random Search and Simulated Annealing have explored the combination of such simple heuristics, generating excellent results compared to the generic MHs. Recent work investigates models that use Artificial Neural Networks that learn from previously generated metaheuristics to enhance their tailoring process and generate MHs with better performance. Such a model proves to have better performance by using Long Short-Term Memory (LSTM) architectures taking advantage of their capabilities to learn patterns on sequences. Nevertheless, the model has the limitation of only processing sequences of numbers.

This work introduces a few hyper-heuristic models based on transformers that can process different representations of metaheuristics. It takes advantage of pre-trained models to only do a fine-tuned process, proving to learn patterns from a set of sequences but does not enhance the tailoring process as well as the state-of-the-art model based on LSTM. Nevertheless, it shows a new path for future works related to transformers and hyper-heuristics for generating modified metaheuristics.

2 Introduction

Optimisation problems have many applications in industry and academia, so they have been studied for a long time. It is also well-known that achieving the optimal solution for an optimisation problem is challenging. Due to that, many approaches work on finding an approximation to the solution [20]. Novel algorithms have been designed and implemented for such optimisation problems, obtaining exceptional results so good that, in some cases, the algorithm can get the optimal solution [14].

Depending on the researcher's interest, an algorithm can be designed to solve a particular instance from a continuous optimisation problem, as is the case of heuristic algorithms. Specifically, one can implement a Metaheuristic (MH), which has a higher level of generalisation, to obtain competitive results for a class of problems. Nevertheless, there is no metaheuristic without mentioning its well-known flexibility and versatility that can always approximate the optimal solution for every optimisation problem, as the No-Free-Lunch theorem states [26]. Researchers and practitioners must know how to select an MH for a given problem in practice, and even then, they must also learn how to set their tuning parameters. Therefore, the question of which metaheuristic is worth implementing to solve a defined problem remains open, and it is commonly answered based on the practitioner expertise or intuition. Many researchers have been working on designing algorithms with high-level abstraction to face such a challenge. One of the proposals is to analyse the metaheuristics as a composition of independent modules (or simple heuristics) using metaphor-less descriptions [3]. This modular structure for metaheuristics can bring different advantages [23]. For example, it allows for systematically analysing MHs according to their components and operators [12]. Moreover, it provides an easy way to configure hybrid metaheuristics, as proposed by Cruz-Duarte *et al.* [11]. Indeed, a research line has recently emerged for investigating algorithms that automatically configure the composition of simple heuristics to generate or modify metaheuristics for a specific problem [18].

Hyper-Heuristics (HHs) are algorithms that search methods or have a learning mechanism for selecting or generating heuristics to solve computational search problems [2]. These techniques evidence the capacity of high-level abstraction and generalisation in tackling problems with different domains. In the last decade, HHs have received great attention, developing many good applications for real-world complex problems [5], *i.e.,* combinatorial optimisation [1, 21] and continuous optimisation [7]. However, it is hard to find many proposals in the latter domain [16]. One example of a unified framework that provides a solver for the AACP is the Customising Optimisation Metaheuristics via Hyper-Heuristic Search (CUSTOMHyS) [11]. This framework provides a methodology for extracting operators from well-known metaheuristics and multiple HH models that explore the heuristic space to generate MH. Moreover, Cruz-Duarte *et al.* reported that the framework tailors metaheuristics with good performances, getting better results than the generic MHs in several problems [7].

One of the most recent progress in this area is the work conducted by Tapia-Avitia *et al.* [25], which contribution is a novel hyper-heuristic model based on neural networks, providing statistical evidence that such a model can generate enhanced metaheuristics with better performance rather than the solvers implemented on the CUSTOMHyS framework. According to their work, one area of opportunity is to integrate other machine learning models for processing the sequences instead of the Long Short-Term Memory architecture proposed to improve the learning process.

The work presented by Tapia-Avitia *et al.* proposed the usage of Long Short-Term Memory (LSTM) architecture because of its capabilities to learn hidden patterns from sequences of numbers. They proved its feasibility and showed that the model has certain learning to identify which simple heuristics are used to enhance a metaheuristic's performance. Nevertheless, the procedure to generate new metaheuristics has a similar abstract structure to the Natural Language Processing (NLP) task of sequence-to-sequence that generate text based on a previous paragraph [27].

The LSTM architecture used on the hyper-heuristic model proposed by Tapia-Avitia *et al.* is capable of considering the order of the elements for a given sequence. Nevertheless, it is limited to learning from sequences of numbers and does not have an attention mechanism. Thus, this work integrates the usage of transformers to use their attention mechanism to potentially improve the learning of hidden patterns from a set of metaheuristics.

This work aims to propose a hyper-heuristic model based on transformers that generates tailoring metaheuristics that produce enhanced solutions for continuous optimisation problems. Besides integrating transformers in a hyper-heuristic model, it contributes to the exploration of a different representation of the metaheuristics.

This document is organised as follows. Section 3 briefly describes the background information about the fundamental concepts of this research. Section 4 explains the proposed approach. Section 5 details the methodology employed to test the experiments. Then, Section 6 presents an analysis of the results, doing statistical tests to make inferences with a certain confidence. Finally, Section 7 wraps up the manuscript with main insights and the paths for future work.

3 Fundamentals

This section details the fundamental concepts mentioned in the introduction. Several of these concepts may seem trivial, but to avoid misunderstandings and controversies, it is established which definitions are considered, *e.g.,* the definition of an optimisation problem, heuristics, and the transformers that are used on the solution model. These concepts are used in this work and are considered to design the solution model.

3.1 Optimisation

Optimisation is one of the most common abstract procedures possible to find in nature. The word "optimum" in Latin means "the ultimate ideal"; similarly, "optimal" means "the best". Then, optimisation can refer to one act, process, or methodology of making a series of decisions to achieve the best results of a problem or event. In this work, the optimisation process is defined as the mathematical model given by a feasible domain and an objective function to minimise, as follows:

Definition 1 (Minimisation problem) *Let $\mathfrak{X} \subseteq \mathfrak{G}$ be a feasible domain, since $\mathfrak{G}$ is an arbitrary domain, and let $f(\vec{x})$ be a real-valued function to be minimised, known as cost function, defined on $\mathfrak{X} \neq 0$ such as $f(\vec{x}) : \vec{x} \in \mathfrak{X} \to \mathbb{R}$. Thus, a minimisation problem represented with the tuple $(\mathfrak{X}, f)$ is stated as*

$$\vec{x_*} = \underset{\vec{x} \in \mathfrak{X}}{\operatorname{argmin}} \{f(\vec{x})\}, \tag{1}$$

where $\vec{x_*} \in \mathfrak{X}$ is the optimal vector (or solution) that minimises the objective function, that is, $f(\vec{x_*}) \leq f(\vec{x}), \forall \vec{x} \in \mathfrak{X}$.

Remark 1 (Sphere Function) *The sphere function has the following equation*

$$f(\vec{x}) = \sum_{i=1}^{D} x_i^2 \tag{2}$$

with $\vec{x} \in \mathbb{R}^D$ where D represents the dimensionality of the problem. The minimisation problem for such a function has the optimal value of 0.

3.2 Heuristics

A heuristic can be interpreted as a sequence of actions [16]. Such a sequence may contain a single or multiple instructions that do not necessarily follow a sequential pattern [11]. According to their level of abstraction, they can be categorised into *simple heuristics*, *metaheuristics*, and *hyper-heuristics*.

First of all, since most heuristics operate over a population (*i.e.,* a set of agents), like those implemented in this work, it is necessary to define it. Thus, a population consists of a finite set of N candidate solutions, which is denoted as $X(t) = \{\vec{x}_1(t), \ldots, \vec{x}_N(t)\}$. When having multiple candidate solutions, one must find a sensible way to pick the best one. To do so, it is considered an arbitrary set of candidate solutions $Z(t)$, which can be designated as, *e.g.,* the entire population ($Z(t) = X(t)$) or the historical evolution of the n-th candidate ($Z(t) = \{\vec{x}_n(0), \ldots, \vec{x}_n(t)\}$). Hence, let $\vec{x}_*(t) \in Z(t)$ be the best position w.r.t. the objective function from $Z(t)$, *i.e.,* $\vec{x}_*(t) = \operatorname{arginf}\{f(Z(t))\}$.

Bearing this information in mind, the heuristics mentioned above are described as follows:

Simple Heuristics (SHs) produce (initialisers, h_i), modify (search operators, h_o), or evaluate a candidate solution (finaliser, h_f) [11]. In that sense, it is noticeable that SHs are the building blocks or primitives for generating more sophisticated methods.

Metaheuristics (MHs) can be defined in general terms as sequences of heuristics. So, an MH can be an iterative procedure that renders an optimal solution for a given optimisation problem. It is common to find MHs with a master strategy, the finaliser, which controls the iterative procedure. However, in this work, it is employed the general model called *unfolded Metaheuristic* (uMH), which delegates the finaliser role to a superior strategy, so it only deal with heterogeneous sequences of heuristics [10].

Hyper-heuristics (HHs) manage low-level heuristics to produce an adequate combination of them to solve the problem effectively instead of manipulating the solutions of the problem to find the optimal solution. So, a HH searches for the optimal heuristic configuration (or metaheuristic) that best approaches the solution of $(\mathfrak{X}, f)$ with the maximal performance $Q_{\max} = Q(\vec{h}_*|\mathfrak{X}, f)$. Consider that $Q(\vec{h}|\mathfrak{X}, f) : H \to \mathbb{R}_+$ is a metric that measures the performance of $\vec{h}$ when it is applied to the problem $(\mathfrak{X}, f)$. In this work, it is *evaluated* an MH running it several times and calculate its performance using the following Q metric:

$$Q(\vec{h}|\mathfrak{X}, f) = -(\texttt{med} + \texttt{iqr})\left(\{\forall\, \vec{x}_{r,*} \in X_* | f(\vec{x}_{r,*})\}\right),$$

where `med` and `iqr` are the median and interquartile range operators applied to a set of fitness values $f(\vec{x}_{r,*})$ obtained from implementing an arbitrary MH $\vec{h}$.

3.3 Transformer

The transformer was originally proposed as a sequence-to-sequence model for machine translation [24]. However, its attention mechanism has achieved state-of-the-art performance on several tasks related to natural language processing [17], computer vision [4], and speech processing [6].

There are a wide variety of transformer models whose usage is recommendable according to their application. According to Lin *et al.* [15], exists three perspectives of these transformer's variants:

1. Model Efficiency. The transformer is inefficient when processing long sequences mainly due to the computation and memory complexity of the self-attention module. Some variants include methods to improve such aspects, such as lightweight attention (*e.g.,* sparse attention variants) and Divide-and-conquer methods (*e.g.,* recurrent and hierarchical mechanism).

2. Model Generalisation. The transformer is a flexible architecture that makes few assumptions about the input data's structural bias, so it is hard to train a model on small datasets. The transformer variants allow to introduce structural bias or regularisation, as well as pre-trained data on large-scale unlabeled data.

3. Model Adaptation. Pre-trained models ready to fine-tune for specific downstream tasks and applications.

3.4 Previous Work

The following articles have proposed hyper-heuristic models for the Automatic Algorithm Composition Problem in recent years.

- The manuscript entitled "*CUSTOMHyS: Customising Optimisation Metaheuristics via Hyper-heuristic Search*" [9] continued the work done in by the previous explained report [7]. The authors implemented a Simulated Annealing approach to explore the heuristic space in this case. Also, they included and expanded the considered collection. They detailed the module implementation of the framework, including a description of how to integrate particular search operators for further investigation, for example, to analyse their performance combined with other simple heuristics or their parameter tuning.

- The article "*Global Optimisation through Hyper-Heuristics: Unfolding Population-Based Metaheuristics*" [10] introduced the definition of unfolded metaheuristics, as well as the categorisation of homogeneous and heterogeneous metaheuristic, extending the possible structures that a metaheuristic could have. In this work, the hyper-heuristic model is based on Simulated Annealing, generates a random sequence of heuristics, and enhances its performance by adding, deleting, swapping, or changing elements of this sequence.

  The results obtained in this paper were better than the metaheuristics from the literature and, in several benchmark functions, better than the tailored metaheuristics from the previous works.

- Furthermore, Cruz-Duarte *et al.* introduced a new version of their previous solvers on "*A Transfer Learning Hyper-heuristic Approach for Automatic Tailoring of Unfolded Population-based Metaheuristics*" [8]. Their main contribution is the integration of transfer-learning techniques that proved to improve the performance of the tailoring process. Their technique uses extra memory to store a transition matrix to help search operators choose according to how often they are used.

- The last contribution on the framework CUSTOMHyS was published by Tapia-Avitia *et al.* "*A Primary Study on Hyper-Heuristics Powered by Artificial Neural Networks for Customising Population-based Metaheuristics in Continuous Optimisation Problems*" [25] that introduces a novel solver based on Neural Network. It compares its performance against basic metaheuristics and unfolded metaheuristics generated by a solver of CUSTOMHyS. The main contribution of this work is to show the learning capabilities of the proposed model to produce metaheuristics with better performance than the implemented solvers in the framework.

It is worth mentioning a key aspect from work done by Tapia-Avitia *et al.* [25]; they represent their generated metaheuristics as a sequence of heuristics, but the such sequence is encoded into a sequence of numbers where each number is respectively its index in the collection of search operators given.

From the perspective of the transformers, such representation of the sequences can be seen as *tokenise* the sequence of heuristics into a sequence of numbers. Bear in mind that such a step of *tokenise* the input data is necessary when the data is pre-processed for the transformers. Nevertheless, tokenisation does not guarantee to work well on the transformer models, as there is no rule or relation in assigning the numbers from a heuristic to its index.

Thus, it can be possible to explore another representation of the sequence of heuristics taking advantage of the fact that transformers receive not only a sequence of numbers but text.

4 Proposal Approach

The proposed model in this work leveraged the sequence generation from Tapia-Avitia *et al.* [25]. The Machine Learning model is used on such generation to classify the sequence into one simple heuristic. It is well-known that training a transformer from scratch is a time-consuming task of several days, which is why it is proposed to use pre-trained models in classification to fine-tune them with training data related to the generation of metaheuristics. This section describes how it is generated the training data for the transformer models, as well as how it is represented the sequence of heuristics.

4.1 Representation of a Sequence of Heuristics

Before explaining the proposal approach, it is necessary to mention how a sequence of heuristics is represented in this work. The CUSTOMHyS framework provides an encoded representation that associates a number to each simple heuristic for a given collection. Then, it is easy to encode and work with a sequence of numbers without losing any information for a given sequence of heuristics. This representation can be widely used on various machine learning models. However, it is possible to exploit the flexibility of the transformer to give it more information when it is being trained.

For example, the following lines are the explicit representation of the simple heuristics in the CUSTOMHyS framework:

```
('genetic_crossover',
 {
    'pairing': 'random',
    'crossover': 'two',
    'mating_pool_factor':
    0.4
 },
 'greedy')

('swarm_dynamic',
 {
    'factor': 1.0,
    'self_conf': 2.54,
    'swarm_conf': 2.56,
    'version': 'constriction',
    'distribution': 'uniform'
 },
 'probabilistic')
```

It contains all the parameters needed to apply such a simple heuristic. Using the encoding representation of numbers will represent the previous simple heuristics as the sequence `66, 196`. To take advantage that transformers can receive text, it is possible to comprise the simple heuristic to give more context about which heuristic is being used. For example, the following representation:

```
GC,r;two;0.4,g

PS,1.0;2.54;2.56;c;u,p
```

Then, there are three possible representations for a sequence of heuristics:

1. Sequence of numbers
2. Sequence of the full description of the simple heuristics
3. Sequence of the compressed representation of the simple heuristics

4.2 Generation of a Sequence

Given an optimisation function and a collection of SHs, it is proposed the following simple procedure to generate a search operator sequence:

1. Initialise an empty unfolded metaheuristic.
2. Use the HH model to select which SH to include next in the current sequence.
3. Add the chosen SH at the end of the sequence.
4. Return to Step 2 if any stopping criterion is not met; otherwise, finish the procedure.

Figure 1 illustrates this procedure. Observe that a new heuristic is added in each iteration. The procedure stops when it reaches the limit of elements established or the chosen simple heuristic no longer improves the performance.

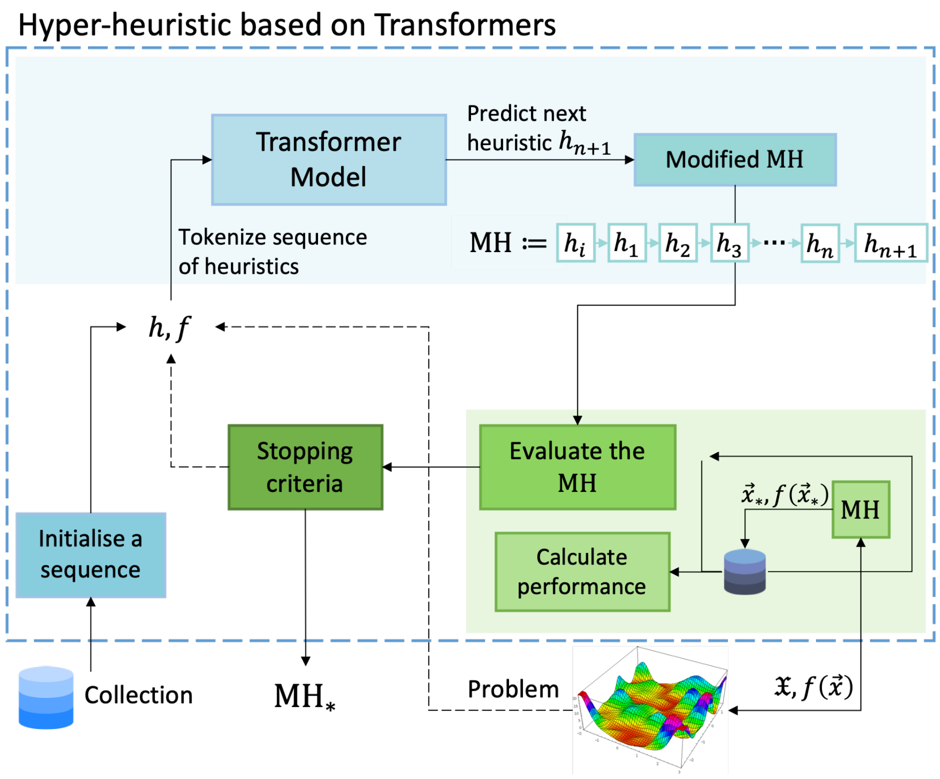
Hyper-heuristic based on Transformers
Transformer Model
Predict next heuristic h_{n+1}
Modified MH
Tokenize sequence of heuristics
MH := h_i h_1 h_2 h_3 ··· h_n h_{n+1}
h, f
Stopping criteria
Evaluate the MH
$\vec{x}_*, f(\vec{x}_*)$
MH
Calculate performance
Initialise a sequence
Collection
MH$_*$
Problem
$\mathfrak{X}, f(\vec{x})$

Figure 1: Data flow of the proposed solution model. Based on: Tapia-Avitia *et al.* [25].

4.3 Training Dataset

In the given context, a training dataset is a collection of search operator sequences that would be used to train a model. The task assigned to the transformer is to classify each sequence into one simple heuristic.

Keeping this in mind, generating a dataset for training the transformer is possible using a given set of search operator sequences. The procedure to obtain such a training dataset is quite simple to describe. First, given a sequence, consider all its prefixes, including the empty prefix, and exclude the prefix corresponding to the whole sequence for a given sequence. Then, each prefix can be associated with the heuristic next in the sequence. Figure 2 depicts such a procedure over a tiny sample. For illustrative purposes, each sequence of heuristics is represented as a sequence of numbers. Such a procedure transforms a sample of uMHs into a training dataset.

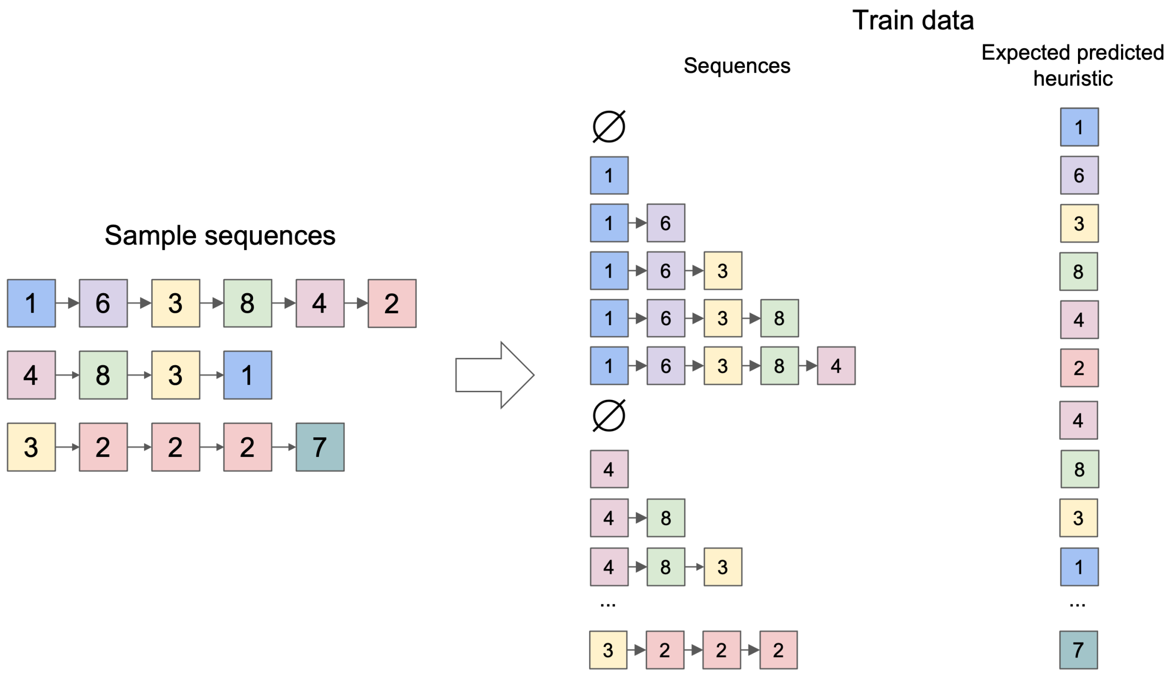

Figure 2: Illustrative example about how to get the training dataset from sample sequences. Source: Tapia-Avitia *et al.* [25].

5 Methodology

In this work, I used Python 3.9.10, a MacBook Pro with M1 Pro and 16 GB RAM, and a macOS Ventura 13.0 system to run the experiments. The implementation of the proposed model uses the open-access framework CUSTOMHyS v1.1, which can be found on `https://github.com/jose-tapia/transformer-hh-ctml` accessed on November 24th, 2022. For fully detailed documentation of version v1.0, please review the previous manuscript published by Cruz-Duarte *et al.* [9]. It is mainly used the modules `Numpy`, `Pandas`, `PyTorch`, and the main models from `HuggingFace` (`https://huggingface.co` accessed on November 24th, 2022).

5.1 Optimisation Function

The optimisation function that is being solved during the experimentation is the `Sphere` with `10` dimensions where the optimal value is `0`. It is used as the low-level domain for the hyper-heuristic models.

5.2 Collection of Simple Heuristics

It leveraged a collection of population-based heuristics from the CUSTOMHyS framework. It is used as the high-level domain of the hyper-heuristic models. This collection has 205 simple heuristics extracted from ten well-known metaheuristics, such as Random Search, Simulated Annealing, Genetic Algorithm, Cuckoo Search, Differential Evolution, Particle Swarm Optimisation, Firefly Algorithm, Stochastic Spiral Optimisation Algorithm, Central Force Optimisation, and Gravitational Search Algorithm. It is considered to use a population of 30 agents for each population-based unfolded metaheuristic.

5.3 Train and Test Dataset of Sequences

The train and test dataset is obtained as follows:

- It is generated a sample of 1,000 metaheuristics using the state-of-the-art model proposed by Tapia-Avitia *et al.* [25], so it is a guarantee that the metaheuristics have great performance. The neural network architecture used on such a model uses an input layer of 60 neurons, two LSTM hidden layers of 20 neurons each using `sigmoid` activation function, and an output layer of 205 neurons. These metaheuristics are limited to up to 100 simple heuristics. Nevertheless, not all metaheuristics reach such a limit. This sample of 1,000 metaheuristic are stored into several files to its public usage. It require 163.7 MB in total to store all the metaheuristics. Such set can be found on `https://github.com/jose-tapia/transformer-hh-ctml/blob/main/stored_sequences.zip` accessed on November 24th, 2022. A few methods to read and generate such metaheuristics are shared in the attached notebook.

- After perform the procedure described on Subsection 4.3, it is obtained a dataset of 94,536 elements.

- It is filtered to use only the 20% of sequences with better performance from this dataset for training to ensure the learning from sequences that can contribute to patterns that could improve the performance.

  Then, the training dataset consists in 18,907 sequences, where each one is labelled with a heuristic that potentially improves the performance of the sequence.

- For the test dataset, the following 20% of the sequences are used.

Bear in mind that the principal purpose of this work is to explore the capabilities of the transformers; the parameters used to generate the dataset are preliminary options selected based on previous experiments to guarantee the generation of good enough sequences.

5.4 Experimentation

It is considered two control variables:

1. Pre-trained model used. The considered pre-trained models are the `distilbert-base-uncased` [22] and `GPT-2` [19]. It is considered the `distilbert` instead of `bert` [13] because it is at least twice faster to train in `distilbert` than `bert`. It is used `GPT-2` instead of `GPT-3` because `GPT-3` is not publicly available yet.

2. The number of epochs to train the model. It is used 1, 2, ..., 10 epochs.

Given a total of 20 transformer models. The models are labelled as follows: models trained from a `distilbert` receive the label `HyBert-`E, where E is the number of epochs used to train a such model. Similarly, models based on `GPT-2` receive the label `HyGpt-`E, where E is the number of epochs used to train the respective model.

The following assumptions are considered: All the models are trained using the same train dataset. The trained models generate metaheuristics to solve the Sphere optimisation problem. All the sequences of simple heuristics are represented as the sequence of numbers.

Each transformer is trained by using the number of epochs assigned, Cross-Entropy as loss function, the AdamW optimiser with a learning rate of 5×10^{-5}, batch size of 32, and a weight decay of 0.01. After training each model, it generates 100 metaheuristics with at most 100 simple heuristics each. Then, the performance is computed using the formula described in Subsection 3.2.

The models are compared to the current state-to-the-art hyper-heuristic model that generates metaheuristics for continuous optimisation problems as well. Such hyper-heuristic model based on neural networks from the recent work presented on Tapia-Avitia *et al.* [25], that uses as architecture an input layer of 60 neurons, two LSTM hidden layers of 20 neurons each using `sigmoid` activation function, and the same output layer of 205 neurons as the trained transformers of this work.

6 Results

6.1 Computational Cost

This work considers pre-trained models to take advantage of, which only requires fine-tuning them in the classification task proposed. Thus, it saves a lot of training time, and only a few epochs are necessary to fine-tune the model. The following costs were performed during the training of the models:

- The memory cost depends on which pre-trained model is used. Models based on `distilbert` need 175 MB to store all their parameters. On the other hand, models based on `GPT-2` required 511 MB.

- The training time per epoch depends on the pre-trained model used as well. By training the models using the train data described, `HyBert`'s models take a constant of 15 minutes per epoch, while `HyGpt`'s models require 1:30 hours per epoch.

- The accuracy in the evaluation dataset for `HyBert`'s models is a constant of 0.009, while the accuracy for `HyGpt`'s models is a constant of 0.005. Then, all the experiments have greater accuracy than the random choice selection $\left(\frac{1}{205} \sim 0.00487\right)$. Even so, `HyBert` surprises with the double of accuracy, but there is no significant change in the accuracy when the model is trained with more epochs.

These statistics do not look promising, as the hyper-heuristic model based on the neural network proposed by Tapia-Avitia *et al.* requires 156 KB to save the model parameters, making it more portable and easy to access to most computers. Such a model takes at most 23 minutes to train the model from scratch with 100 epochs and achieves an accuracy of at least 3.6 and, at most, seven times better than the random choice probability.

6.2 Performance

Bear in mind that the main objective of this work is to analyse the performance of the hyper-heuristics based on transformers. It generated 100 metaheuristics from each one of the twenty trained models and used the following metric to compute the model's performance:

$$Q(\vec{h}|\mathfrak{X}, f) = -(\texttt{med} + \texttt{iqr})\left(\{\forall\, \vec{x}_{r,*} \in X_* | f(\vec{x}_{r,*})\}\right)$$

Figure 3 illustrate the performance evolution of the models through the number of epochs used to train the models. Notice that there is no apparent improvement in the performance when the number of epochs increases. Nevertheless, it is worth mentioning that those models based on `distilbert`, `HyBert`, have a consistent performance in comparison to the models based on `GPT-2`.

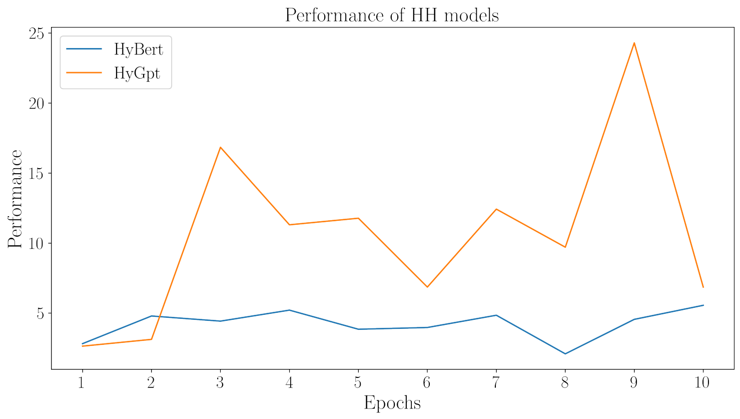

Figure 3: Performance evolution through the number of epochs on the pre-trained models.

The model that obtained the best performance from the experiments is achieved by `HyBert-8` with a value of 2.0951. Comparing the values obtained by `HyGpt`, it can look like a good performance. Nevertheless, the state-of-the-art model based on neural networks obtained a performance of 0.08057, significantly less than the value reported in this work.

7 Conclusions

The usage of transformer models sounded promising, as the LSTM models obtained good results, and the addition of an attention mechanism could improve the learning of hidden patterns in the sequence of heuristics. Nevertheless, it was not the case, at least by using pre-trained models and using at most 10 epochs on the `distilbert` and `GPT-2` models.

This work have the following contributions:

- Conceptualise how can be integrated transformer models in the generation of metaheuristics.
- Preliminary results of the usage of pre-trained transformer models.

- Public implementation and reproducible results of the proposed models (GitHub: https://github.com/jose-tapia/transformer-hh-ctml accessed on November 24th, 2022).

- Published models in HuggingFace hub. The name of the models can be found in the attached notebook. HuggingFace profile: https://huggingface.co/josetapia accessed on November 24th, 2022.

7.1 Future Work

Besides the unexpected results, it is possible to extend this work and change possible pitfalls.

- The main problem could be the consideration of a relatively small dataset for training. To surmount this issue, it is possible to use another pre-trained model or a different method of training that not only receives the data but the structural bias of it to be able to train the transformers with small datasets [15].

  Another approach could be to increase the sample of metaheuristics used to generate the training dataset.

- The second major problem is that the transformers can receive data and learn to classify it, but at this moment, the `HuggingFace`'s transformers models does not support a new parameter to receive a vector with the weight's importance of each element in the sample, so the model can know which samples should receive priority to learn first.

  The hyper-heuristic model based on a neural network from Tapia-Avitia *et al.* take advantage of such features in their Tensorflow models.

- It is possible to extend the training session of the models to take not only half of the day training but several days to see if it is possible to improve the accuracy or the performance of the model.

7.2 Additional Results

It was considered the usage of the other representations proposed in this work as the full description of the simple heuristics or its comprised version. These ideas were discarded for this project because it was necessary to deal with a sequence of tokens of length in the order of 10,000, or the training time of the sequence was near to 6 hours per epoch and did not have outstanding performance as the models presented on Section 6.

It was considered to implement a modified Categorical-Cross Entropy loss function that takes each sequence's fitness value to prioritise those with better performance. Nevertheless, the time required to train an epoch was nearly 5 hours using the straightforward representation of a sequence of numbers. Thus, it was given priority to faster models and left this exploration on the second plane.

Another considered approach is using the transformer model not as a classifier but as a text generator. Then, it could learn the abstract representation of the sequences to predict new unseen simple heuristics. For this approach, it was considered the same training dataset used for the classification models, and the time per epoch was nearly 50 minutes. It was trained by 15 epochs and resulted in the learning of part of the representation of the simple heuristic. This approach is not fully functional, as the transformer model only sometimes generates the parameters of the simple heuristics correctly. Further improvements can be made to fix this issue, using a self-correction mechanism or giving the structural bias to the transformer model to enhance the generation.

References

[1] Amaya, I., Ortiz-Bayliss, J. C., Rosales-Perez, A., Gutierrez-Rodriguez, A. E., Conant-Pablos, S. E., Terashima-Marin, H., and Coello, C. A. C. Enhancing selection hyper-heuristics via feature transformations. *IEEE Computational Intelligence Magazine 13*, 2 (2018), 30–41.

[2] Burke, E. K., Hyde, M. R., Kendall, G., Ochoa, G., Özcan, E., and Woodward, J. R. A classification of hyper-heuristic approaches: revisited. In *Handbook of Metaheuristics*. Springer, 2019, pp. 453–477.

[3] Campelo, F., and Aranha, C. Sharks, zombies and volleyball: Lessons from the evolutionary computation bestiary.

[4] Carion, N., Massa, F., Synnaeve, G., Usunier, N., Kirillov, A., and Zagoruyko, S. End-to-end object detection with transformers. In *European conference on computer vision* (2020), Springer, pp. 213–229.

[5] Chakhlevitch, K., and Cowling, P. Hyperheuristics: recent developments. In *Adaptive and multilevel metaheuristics*. Springer, 2008, pp. 3–29.

[6] Chen, X., Wu, Y., Wang, Z., Liu, S., and Li, J. Developing real-time streaming transformer transducer for speech recognition on large-scale dataset. In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (2021), IEEE, pp. 5904–5908.

[7] Cruz-Duarte, J. M., Amaya, I., Ortiz-Bayliss, J. C., Conant-Pablos, S. E., and Terashima-Marín, H. A primary study on hyper-heuristics to customise metaheuristics for continuous optimisation. In *2020 IEEE Congress on Evolutionary Computation (CEC)* (2020), IEEE, pp. 1–8.

[8] Cruz-Duarte, J. M., Amaya, I., Ortiz-Bayliss, J. C., and Pillay, N. A transfer learning hyper-heuristic approach for automatic tailoring of unfolded population-based metaheuristics. In *2022 IEEE Congress on Evolutionary Computation (CEC)* (2022), IEEE.

[9] Cruz-Duarte, J. M., Amaya, I., Ortiz-Bayliss, J. C., Terashima-Marín, H., and Shi, Y. CUSTOMHyS: Customising Optimisation Metaheuristics via Hyper-heuristic Search. *SoftwareX 12* (2020), 100628.

[10] Cruz-Duarte, J. M., Ortiz-Bayliss, J. C., Amaya, I., and Pillay, N. Global Optimisation through Hyper-Heuristics: Unfolding Population-Based Metaheuristics. *Appl. Sci. 11*, 12 (jun 2021), 5620.

[11] Cruz-Duarte, J. M., Ortiz-Bayliss, J. C., Amaya, I., Shi, Y., Terashima-Marín, H., and Pillay, N. Towards a Generalised Metaheuristic Model for Continuous Optimisation Problems. *Mathematics 8*, 11 (2020), 2046.

[12] de Armas, J., Lalla-Ruiz, E., Tilahun, S. L., and Voss, S. Similarity in metaheuristics: a gentle step towards a comparison methodology. *Natural Computing* (2021), 1–23.

[13] Devlin, J., Chang, M., Lee, K., and Toutanova, K. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR abs/1810.04805* (2018).

[14] Ezugwu, A. E., Shukla, A. K., Nath, R., Akinyelu, A. A., Agushaka, J. O., Chiroma, H., and Muhuri, P. K. Metaheuristics: a comprehensive overview and classification along with bibliometric analysis. *Artificial Intelligence Review 54*, 6 (2021), 4237–4316.

[15] Lin, T., Wang, Y., Liu, X., and Qiu, X. A survey of transformers. *AI Open* (2022).

[16] Pillay, N., and Qu, R. *Hyper-Heuristics: Theory and Applications*. Springer, 2018.

[17] Qiu, X., Sun, T., Xu, Y., Shao, Y., Dai, N., and Huang, X. Pre-trained models for natural language processing: A survey. *Science China Technological Sciences 63*, 10 (2020), 1872–1897.

[18] Qu, R., Kendall, G., and Pillay, N. The general combinatorial optimization problem: Towards automated algorithm design. *IEEE Computational Intelligence Magazine 15*, 2 (2020), 14–23.

[19] Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. Language models are unsupervised multitask learners.

[20] Ramirez, A., Romero, J. R., and Ventura, S. A survey of many-objective optimisation in search-based software engineering. *Journal of Systems and Software 149* (2019), 382–395.

[21] Sánchez, M., Cruz-Duarte, J. M., carlos Ortíz-Bayliss, J., Ceballos, H., Terashima-Marin, H., and Amaya, I. A systematic review of hyper-heuristics on combinatorial optimization problems. *IEEE Access 8* (2020), 128068–128095.

[22] Sanh, V., Debut, L., Chaumond, J., and Wolf, T. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *ArXiv abs/1910.01108* (2019).

[23] Stegherr, H., and Hähner, J. Analysing metaheuristic components.

[24] Sutskever, I., Vinyals, O., and Le, Q. V. Sequence to sequence learning with neural networks. *Advances in neural information processing systems 27* (2014).

[25] Tapia-Avitia, J. M., Cruz-Duarte, J. M., Amaya, I., Ortiz-Bayliss, J. C., Terashima-Marin, H., and Pillay, N. A primary study on hyper-heuristics powered by artificial neural networks for customising population-based metaheuristics in continuous optimisation problems. In *2022 IEEE Congress on Evolutionary Computation (CEC)* (2022), IEEE, pp. 1–8.

[26] Wolpert, D. H., and Macready, W. G. No free lunch theorems for optimization. *IEEE transactions on evolutionary computation 1*, 1 (1997), 67–82.

[27] Yun, C., Bhojanapalli, S., Rawat, A. S., Reddi, S. J., and Kumar, S. Are transformers universal approximators of sequence-to-sequence functions? *arXiv preprint arXiv:1912.10077* (2019).